Fonctions et procédures

Sous-algorithmes

Introduction

- Les algorithmes que nous avons étudiés étaient constitués d'un seul bloc (Bloc principal),
- Il arrive fréquemment qu'on utilise des traitements similaires plusieurs fois à des endroits différents.
- A chaque fois on est alors obligé de répéter la séquence d'instructions qui se rapportent à ces traitements.

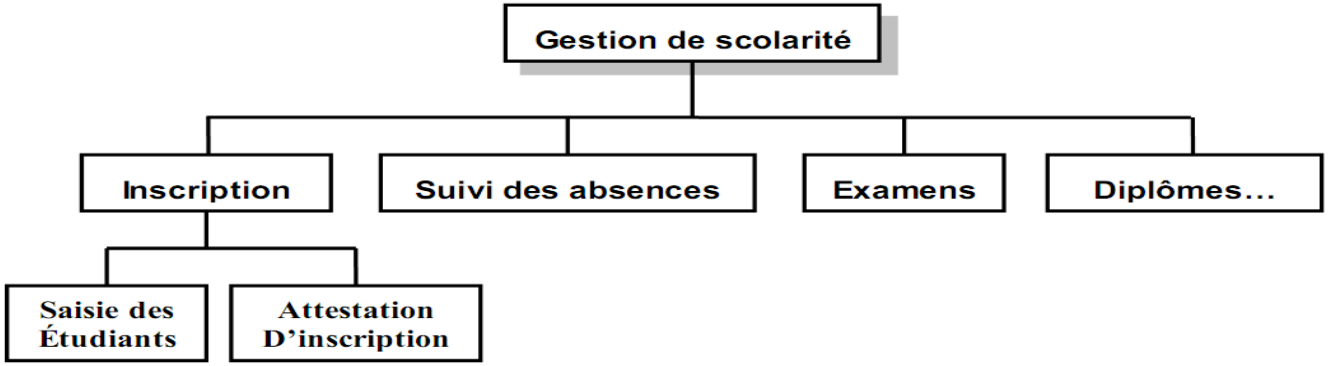
Introduction

Pour une bonne conception d'un algorithme:

- **Il est préférable de décomposer le problème en mini-blocs** (Tâches indépendantes ou modules) ayant chacun un rôle bien précis,
- et qui à leur tour peuvent être décomposées jusqu'à obtenir des tâches simple à résoudre.
- Ces mini-blocs sont des sous-algorithmes.

Introduction

A titre d'exemple, un algorithme de gestion de scolarité peut être découper en plusieurs modules : inscription, suivi des absences, examens, diplômes, etc.



Introduction

- En algorithmique il existe deux types de sous-algorithmes:

1. **Les procédure:** Une procédure est une instruction non élémentaire ayant un nombre quelconque de paramètres en entrée et en sortie

Exemple de procédure: l'instruction échange de deux nombres entier A et B est une instruction non élémentaire pour le processeur et doit être expliciter par le concepteur en écrivant une procédure

Introduction

2. **Les fonctions:** Une fonction est une tâche qui réalise un calcule non élémentaire et produit une valeur unique:

Exemple de fonction: la tâche faire la moyenne de deux nombres se fait par un calcule et produit une valeur unique (une moyenne qui est une valeur réel). Elle doit être spécifier par une fonction.

Un sous-algorithme est obligatoirement caractérisé par un nom (un identifiant) unique

Structure d’un algorithme

- Un algorithme doit suivre la structure suivante:

```
Algorithme  nom_algo;
Const /*Définition des constantes*/
Type /* Définition des types*/
Var /* Définition des variables globales*/
/*Définition des sous-programmes: fonctions et procédures*/
Début
        /* Instructions de l’algorithme principal*/
Fin.
```

Exemple de fonction (sans argument)

```
Algorithme essai;
Var
    I, Som : entier;
Fonction Somme (): entier;
    Var
        S: entier;
    Debut /*Début de la fonction*/
            S ← 0 ;
            Pour I ← 1  a 100 Faire
                        S ← S + i
            FinPour ;
            Somme ← S; // retourner S;
    Fin; /*Fin de la fonction*/
```

```
Debut /*Début de l'algorithme*/
        Som ← Somme();
        Ecrire ('La somme des 100 premiers nombres est', S);
Fin. /*Fin de l'algorithme*/
```

Fonction qui calcule la somme des 100 premiers nombres

Exemple de fonction (avec argument)

```
Algorithme essai;
Var
    I, Som, n : entier;
Fonction Somme (n: entier): entier;
    Var
        S: entier;
    Debut /*Début de la fonction*/
            S ← 0 ;
            Pour I ← 1  a n Faire
                        S ← S + i
            FinPour ;
            Somme ← S;
    Fin; /*Fin de la fonction*/
```

```
Debut /*Début de l'algorithme*/
        lire(n);
        Som ← Somme(n);
        Ecrire ('La somme des', n, 'premiers nombres est', S);
Fin. /*Fin de l'algorithme*/
```

Fonction qui calcule la somme des n premiers nombres (n donnée par l'utilisateur

Exemple de procédure (sans argument)

```
Algorithme essai;
Var
    I, S : entier;
Procedure Somme ()
    Debut /*Début de la Procédure*/
            S ← 0 ;
            Pour I ← 1  a 100 Faire
                        S ← S + I;
            FinPour ;
            Ecrire ('La somme des 100 premiers nombres est', S);
     Fin; /*Fin de la Procédure*/
Debut /*Début de l'algorithme*/
            Somme();
Fin. /*Fin de l'algorithme*/
```

Exemple de procédure (avec argument)

```
Algorithme essai;
Var
    I, som, A : entier;
Procedure Somme (E:n: entier, S: s:entier)
    Debut /*Début de la Procédure*/
            s ← 0 ;
            Pour I ← 1 a n Faire
                        s ← s + I;
            FinPour ;
                Fin; /*Fin de la Procédure*/
Debut /*Début de l'algorithme*/
            lire (A);
            Somme(A,som);
            Ecrire ('La somme des', n,' premiers nombres est', som);
Fin. /*Fin de l'algorithme*/
```

Description d’un sous-algorithme

- Un S-A est décomposée de 3 parties:
  1. Un en-tête.
  2. Une partie déclaration locale.
  3. Une partie instruction.

```
Procedure Somme (n: entier)

Var
        a,b:entier;

Debut /*Début de la Procédure*/
        lire(a,b);
        ,,,,,
        ,,,
Fin; /*Fin de la Procédure*/
```

Description d’une tâche

- Représentation de l’en-tête :

a. En-tête de procédure:

Procédure nom-procédure (paramètres-formels)

Paramètres-formels = paramètres de la procédure

Exemple:

Procédure ECHANGE(<u>ES</u> A,B: entier)

Description d’une tâche

- Représentation de l’en-tête :

b. En-tête de fonction:

Fonction nom-fonction (paramètres-formels): type-fonction

Paramètres-formels = paramètres de la fonction

Type-fonction = type du résultat retourné par la fonction

Exemple:

fonction MOYENNE (**E** X,Y: réel): réel

1. Description d'une tâche

- Représentation de l'en-tête :
  - Commence par l'un des mots clés: « **procédure** » ou « **fonction** »,
  - Suivi du nom de la tâche, (exemple: « Somme »)
  - Puis par une liste de paramètre formels, (exemple : « n:entier »)

Un paramètre formel est une variable définit dans l'en-tête et utilisée dans la partie instruction. Il est défini par son rôle:

- **Donnée** (s'il s'agit d'un paramètre d'entrée),
- **Résultat** (si='il s'agit de paramètre de sortie)
- **Donnée-résultat** (s'il s'agit de paramètre servant d'entrée et de sortie à la fois)

1. En-tête

- Représentation des paramètre formelle:

En LA:

- **E** définit un paramètre donnée,
- **S** définit un paramètre résultat,
- et **ES** (**VAR**) définit un paramètre donnée-résultat

E, **S** et **ES** sont soulignés: ce sont des mots-clés

Exemple de procédure

Procédure ECHANGE (**<u>ES</u>** A,B: entier)
Var
  C: entier;
Début
  C ← B;
  B ← A;
  A ← C;
Fin

Exemple de fonction

Fonction MOYENNE (**E** X,Y réel): réel

Début

MOYENNE ← (X+ Y)/2;

Fin.

Affectation du résultat de la fonction au nom de la fonction

Remarque: La partie déclaration locale peut être vide.

Déclaration d’un tâche

- Une tâche est vue comme un objet par l’algorithme qui l’utilise. Elle doit donc être déclarer dans la partie déclaration de ce dernier.

a. Cas d’une procédure

<u>Procédure</u>

ECHANGE

b. Cas d’une fonction

<u>fonction</u>

MOYENNE: réel

Appel d’une tâche

- Pour appeler une procédure, il suffit d’écrire le nom de cette procédure.
- **Exemple:**

```
ALGORITHME PROC
CONST
        titre = "Algorithmique"

 Procédure TRAIT( )
 VAR
        i : entier
 DÉBUT
        Pour i de 1 à 13 faire
                Écrire ( " - " )
        FinPour
 FIN

DÉBUT(Programme principal)
        Ecrire (titre)
        TRAIT( )        /* Appel de la fonction TRAIT*/
FIN
```

Algorithmique

Généralité

- Dans le bloc principal il suffit d’écrire le nom du sous-algorithme pour déclencher le calcul.
- La liste des paramètres est facultative. Mais quand elle existe, ces paramètres sont déclarés de la même façon qu’on déclare une série de variables de différents types.
- La portée d’une variable est l’ensemble des sous-algorithmes où cette variable est connue
- Une variable globale est une variable définie dans l’en-tête de l’algorithme principal. Elle est utilisable dans n’importe quel sous-algorithme sans nécessité de redéfinition. Donc sa portée est totale

Généralité

- Lorsque le nom d’une variable locale est identique à une variable globale, la variable globale dans ce sous-algorithme devient inaccessible
- Une variable locale est une variable définie à l’intérieur d’un sous-algorithme. Cette variable est inutilisable à l’extérieur du bloc. Il serait donc erroné de l’utiliser dans le bloc principal ou dans un autre sous-bloc appartenant à l’algorithme.
- Un paramètre d’un sous-algorithme est une variable locale particulière qui est associée à une variable ou constante
- Un paramètre est une valeur du bloc principal dont le sous-algorithme a besoin pour exécuter avec des données réelles l’enchaînement d’actions qu’il est chargé d’effectuer.

Les paramètres

Considéré comme une variable locale, un paramètre est une valeur du bloc principal dont la procédure a besoin pour exécuter avec des données réelles l'enchaînement d'actions qu'il est chargé d'effectuer. On distingue 2 types de paramètres :

- **Les paramètres formels (fictifs)** qui sont les paramètres qui figurent dans l'entête de la déclaration de la procédure, et sont utilisés dans les instructions de la procédure. On déclare les paramètres formels pendant la déclaration du procédure.
- **Les paramètres effectifs (réels)** qui sont des valeurs réelles (constantes ou variables) reçues par la procédure au cours de l'exécution du bloc principal, et qui figurent dans l'instruction d'appel de la procédure et sont substitués aux paramètres formels au moment de l'appel de la procédure.

Les paramètres

- Lorsque la déclaration d’une procédure comporte des paramètres formels, ceux-ci doivent être représentés chacun par son identificateur ainsi que par son type.

- Ainsi pendant la construction de l’algorithme principal, il faudra toujours vérifier à ce que chaque appel du procédure soit suivi d’une liste de paramètres effectifs correspondant (en nombre, rang, et type) à la liste des paramètres formels.

Les paramètres

- **Modes de passage des paramètres:**

  Lorsque l'on conçoit un algorithme, on a juste besoin de spécifier, pour chaque paramètre:

  - **s'il est donné en entrée,**
  - **s'il est calculé en sortie,**
  - **ou s'il est donné en entrée puis modifié par l'algorithme pour être ensuite retourné en sortie.**

  Ainsi, au niveau de l'algorithme, on spécifie pour chaque paramètre son mode (entrée, sortie ou entrée/sortie).

Les paramètres

- Au moment de programmer l'algorithme, il s'agit de trouver un mécanisme permettant d'implémenter le comportement correspondant au mode choisi :

1) pour un paramètre en entrée : lors de l'appel de la procédure, il faut passer la valeur du paramètre effectif au paramètre formel ; pendant l'exécution de la procédure appelée, si le paramètre formel est modifié, il ne faut pas répercuter cette modification sur le paramètre effectif.

2) pour un paramètre en sortie : à la fin de l'exécution de la procédure appelée, il faut transmettre la valeur du paramètre en sortie au paramètre effectif correspondant.

Les paramètres

3) pour un paramètre en entrée/sortie : lors de l'appel de la procédure, il faut passer la valeur du paramètre effectif au paramètre formel ; à la fin de l'exécution de la procédure appelée, il faut transmettre la nouvelle valeur du paramètre formel au paramètre effectif correspondant.

- Pour implémenter ces comportements, on dispose de deux mécanismes de passage de paramètres : le passage par valeur et le passage par référence (variable).

a) un paramètre en entrée sera généralement passé par valeur.

b) un paramètre en sortie ou en entrée/sortie sera nécessairement passé par référence ;

Les paramètres

Cependant, lorsque le paramètre en entrée est codé sur un grand nombre d'octets (ce sera le cas des tableaux par exemple), on ne le passera pas par valeur mais par référence afin d'éviter d'avoir à recopier sa valeur, ce qui peut être long et couteux en espace mémoire.

I. Exemple de passage par valeur:

```
ALGORITHMEPASSAGE_VALEUR
VAR
        x : entier
```

```
Procédure ESSAI ( i : entier )
DÉBUT
i ← 3 * i
Écrire ( "Le paramètre valeur i =" , i )
FIN
```

```
DÉBUT (P.P)
        Ecrire ( "Donner la valeur de X : " )
        Lire (x)
        Ecrire ( "Avant appel x = " , x )
        ESSAI ( x )                    /* Appel de la fonction
        Ecrire ( "Après appel x = " , x )
FIN
```

Le passage de paramètres par valeur permet au programme appelant de transmettre une valeur à la procédure appelée. Le paramètre formel « i » est une variable simple. Le paramètre effectif correspondant « x » est une expression de même type que le paramètre formel.

I. Exemple de passage par valeur:

```
ALGORITHMEPASSAGE_VALEUR
VAR
        x : entier
```

```
Procédure ESSAI ( i : entier )
DÉBUT
i ← 3 * i
Écrire ( "Le paramètre valeur i =" , i )
FIN
```

```
DÉBUT (P.P)
        Ecrire ( "Donner la valeur de X : " )
        Lire (x)
        Ecrire ( "Avant appel x = " , x )
        ESSAI ( x )                    /* Appel de la fonction
        Ecrire ( "Après appel x = " , x )
FIN
```

Avant d'exécuter les instructions de la procédure la valeur du paramètre effectif est affectée au paramètre formel. Dans ce cas, le paramètre formel est considéré comme une variable locale qui est initialisée lors de l'appel.

```
ALGORITHMEPASSAGE_VALEUR
VAR
        x : entier
```

```
Procédure ESSAI ( i : entier )
DÉBUT
i ← 3 * i
Écrire ( "Le paramètre valeur i =" , i )
FIN
```

```
DÉBUT (P.P)
        Ecrire ( "Donner la valeur de X : " )
        Lire (x)
        Ecrire ( "Avant appel x = " , x )
        ESSAI ( x )                    /* Appel de la fonction
        Ecrire ( "Après appel x = " , x )
FIN
```

- Avant appel x =10
- Après appel x =10

x = 10	P.P	Procédure
Appel de « ESSAI »	x = 10	i = 10
Exécution de « ESSAI »	x = 10	i = 30
Après « ESSAI »	x = 10	i = 30

Le transfert d'information est effectué dans un seul sens, du programme principal vers la procédure.

Sens de transfert :

P.P. ⟶ Procédure

Toute modification du paramètre formel est sans conséquent sur le paramètre effectif.

I. Exemple de passage par référence:

```
ALGORITHME PASSAGE_VARIABLE
VAR
        x : entier

Procédure ESSAI ( VAR i : entier )
DÉBUT
 i ← 3 * i
Écrire ( "Le paramètre variable i =" , i )
FIN

DÉBUT(P.P)
        Ecrire ( "Donner la valeur de X : " )
        Lire (x)
        ESSAI ( x )                    /* Appel
        Ecrire ( "Après appel x = " , x )
FIN
```

Le passage de paramètres par variable permet au programme appelant de transmettre une valeur à la procédure appelée et vice-versa. Le paramètre formel « i » est une variable. Le paramètre effectif correspondant « x » doit également être une variable du même type que le paramètre formel.

I. Exemple de passage par référence:

```
ALGORITHME PASSAGE_VARIABLE
VAR
        x : entier

Procédure ESSAI ( VAR i : entier )
DÉBUT
 i ← 3 * i
Écrire ( "Le paramètre variable i =" , i )
FIN

DÉBUT(P.P)
        Ecrire ( "Donner la valeur de X : " )
        Lire (x)
        ESSAI ( x )                    /* Appel
        Ecrire ( "Après appel x = " , x )
FIN
```

Pendant l'exécution des instructions de la procédure, le paramètre formel fait référence au même contenant variable que celui désigné par le paramètre effectif.

ALGORITHME PASSAGE_VARIABLE

VAR

x : entier

Procédure ESSAI (VAR i : entier)
DÉBUT

i ← 3 * i
Écrire ("Le paramètre variable i =" , i)

FIN

DÉBUT(P.P)

Ecrire ("Donner la valeur de X : ")
Lire (x)
ESSAI (x) /* *Appel*
Ecrire ("Après appel x = " , x)

FIN

- Avant appel x =10
- Après appel x =30

x = 10	P.P	Procédure
Appel de « ESSAI »	x = 10	i = 10
Exécution de « ESSAI »	x = 30	i = 30
Après « ESSAI »	x = 30	i = 30

Le transfert d'information est effectué dans les deux sens, du programme principal vers la procédure et vice-versa.

Sens de transfert :

P.P. $\rightleftarrows$ **Procédure**

Toute modification du paramètre formel entraîne automatiquement la modification de la valeur du paramètre effectif.

Exemple:

```
ALGORITHME PARAMÈTRE
VAR
        i, j : entier
Procédure TRANSMISSION ( a : entier ; VAR b : entier )
DÉBUT
a ← a + 100
  b ← b + 100
Écrire ( " a =" , a , " b =" , b)
FIN

DÉBUT(P.P)
        i ← 1
        j ← 2
        TRANSMISSION ( i , j )
        Écrire ( " i =" , i , " j =" , j)
FIN
```

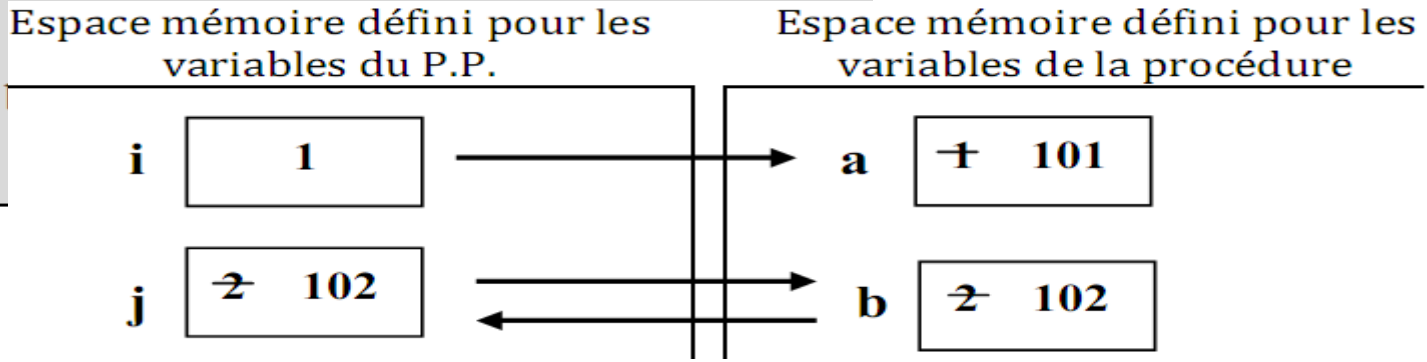
i	1	
j	2 102	
a	1 101	
b	2 102	
Résultat	**a = 101**	**b = 102**
	i = 1	**j = 102**

Exemple:

ALGORITHME PARAMÈTRE
VAR
i, j : entier

Procédure TRANSMISSION (a : entier ; VAR b : entier)
DÉBUT
a ← a + 100
b ← b + 100
Écrire (" a =" , a , "
FIN

DÉBUT(P.P)
i ← 1
j ← 2
TRANSMISSION (i , j)
Écrire (" i =" , i , " j =" , j)
FIN



FIN